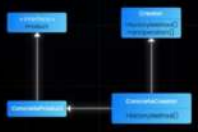
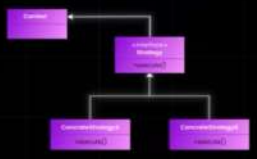
Factory Method



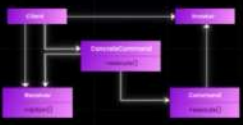
Observer

Strategy



Decorator

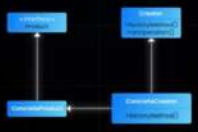
Command



Adapter

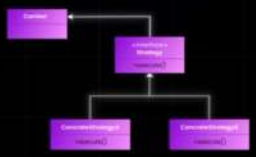
BYTEMONK

Factory Method



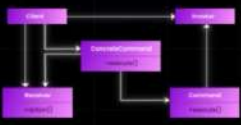
Observer

Strategy



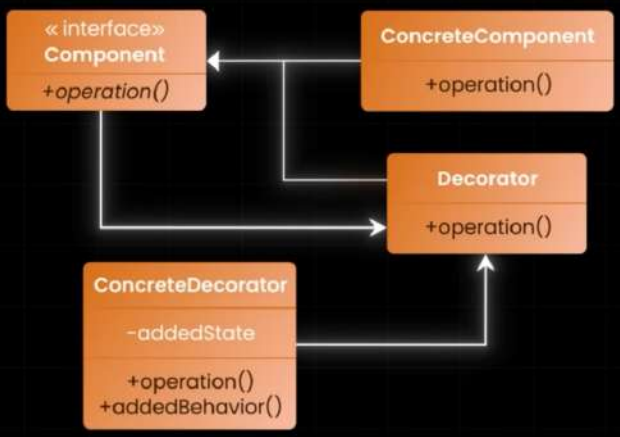
Decorator

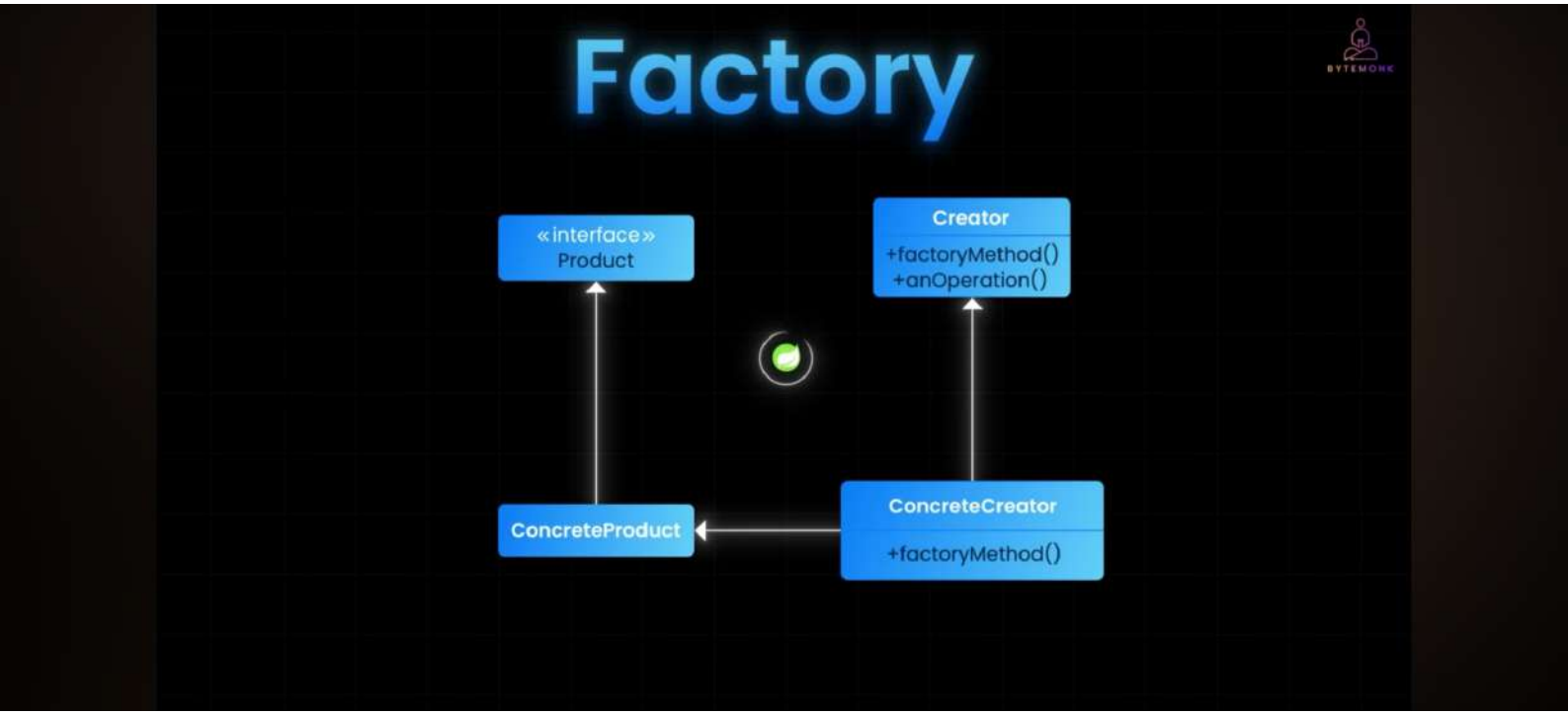
Command



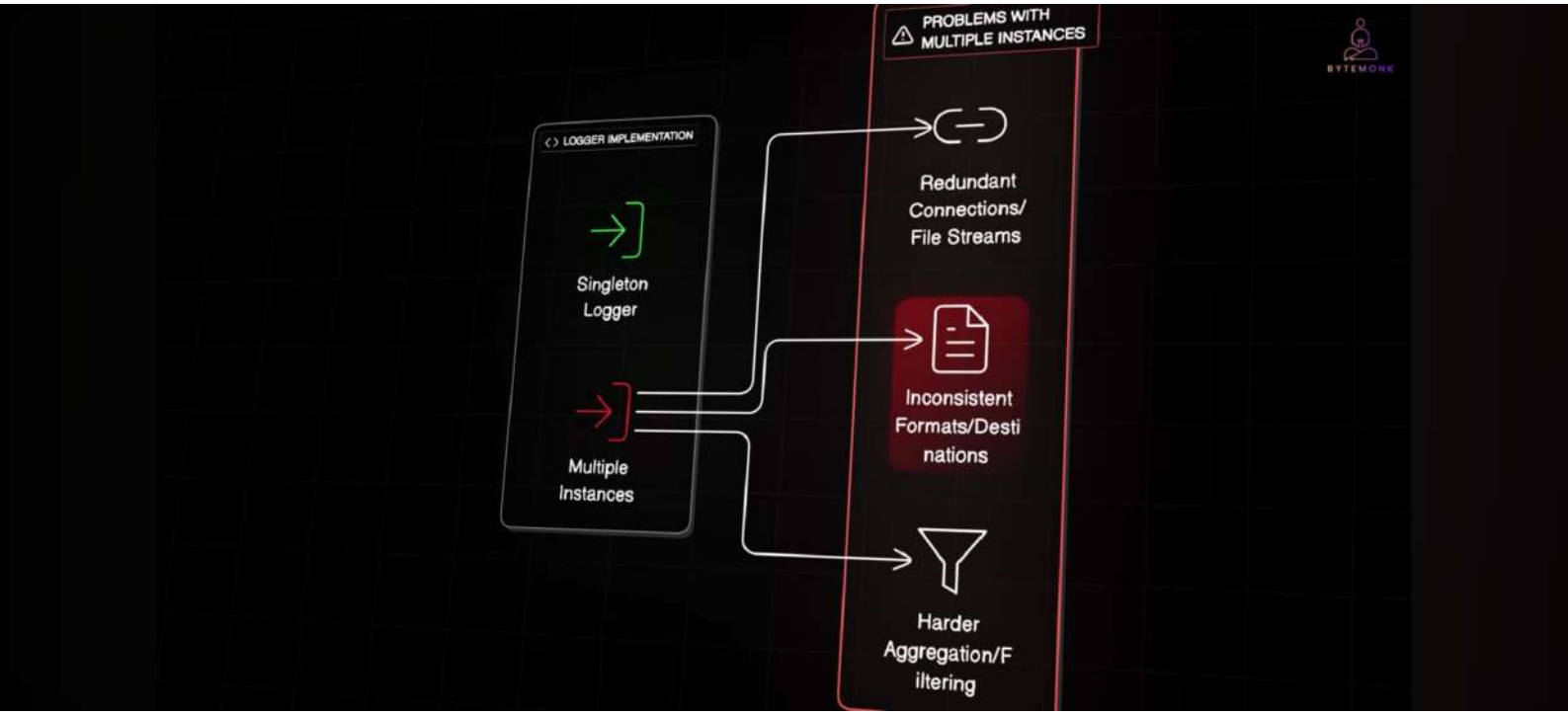
BYTEMONK

Decorator


« interface»
Component
+operation()
ConcreteComponent
+operation()
Decorator
+operation()
ConcreteDecorator
-addedState
+operation()
+addedBehavior()


Factory
BYTEMONK
«interface»
Product
Creator
+factoryMethod()
+anOperation()
ConcreteProduct
ConcreteCreator
+factoryMethod()


BYTEMONK
Shared Vocabulary
C Abstract Factory
S Adapter
S Bridge
C Builder
B Chain of Responsibility
B Command
S Composite
S Decorator
S Facade
C Factory Method
S Flyweight
B Interpreter
B Iterator
B Mediator
B Memento
C Prototype
S Proxy
B Observer
C Singleton
B State
B Strategy
B Template Method
B Visitor


PROBLEMS WITH MULTIPLE INSTANCES
BYTEMONK
LOGGER IMPLEMENTATION
Singleton Logger
Multiple Instances
Redundant Connections/ File Streams
Inconsistent Formats/Desti nations
Harder Aggregation/F iltering

Singleton

Logging - Configuration - Environment setup

- It hides dependencies—makes unit testing harder
- It introduces a global state, which can lead to unexpected side effects
- And overusing it can violate the Single Responsibility Principle

Singleton

- It hides dependencies—makes unit testing harder
- It introduces a global state, which can lead to unexpected side effects
- And overusing it can violate the Single Responsibility Principle

Singleton

- It hides dependencies—makes unit testing harder
- It introduces a global state, which can lead to unexpected side effects

Singleton

- It hides dependencies—makes unit testing harder


BYTEMONK
Singleton
Anti Pattern


BYTEMONK
Singleton

Lazy Initialization

BYTEMONK

java

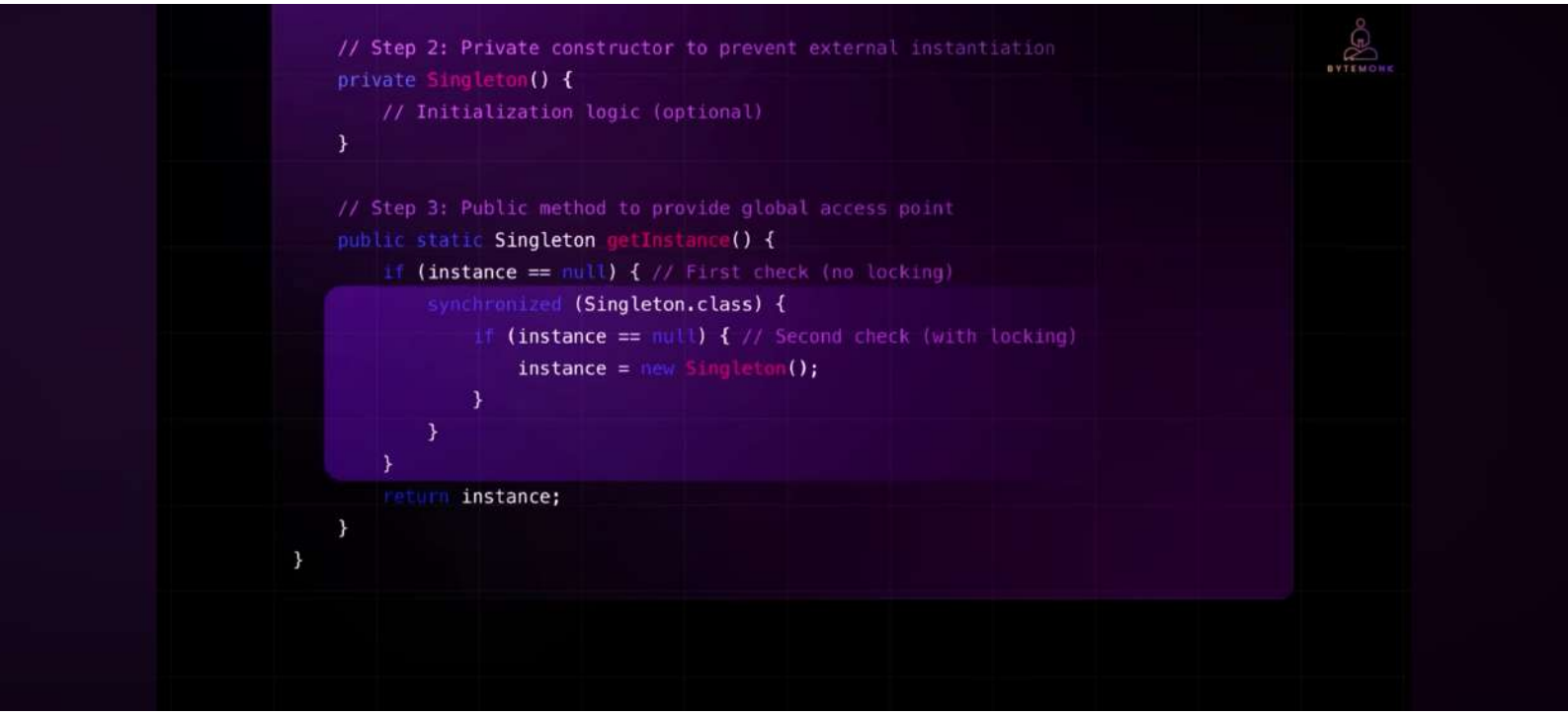
```java
public class Singleton {

    // Step 1: Volatile instance to prevent instruction reordering
    private static volatile Singleton instance = null;

    // Step 2: Private constructor to prevent external instantiation
    private Singleton() {
        // Initialization logic (optional)
    }

    // Step 3: Public method to provide global access point
    public static Singleton getInstance() {
        if (instance == null) { // First check (no locking)
            synchronized (Singleton.class) {
                if (instance == null) { // Second check (with locking)
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```

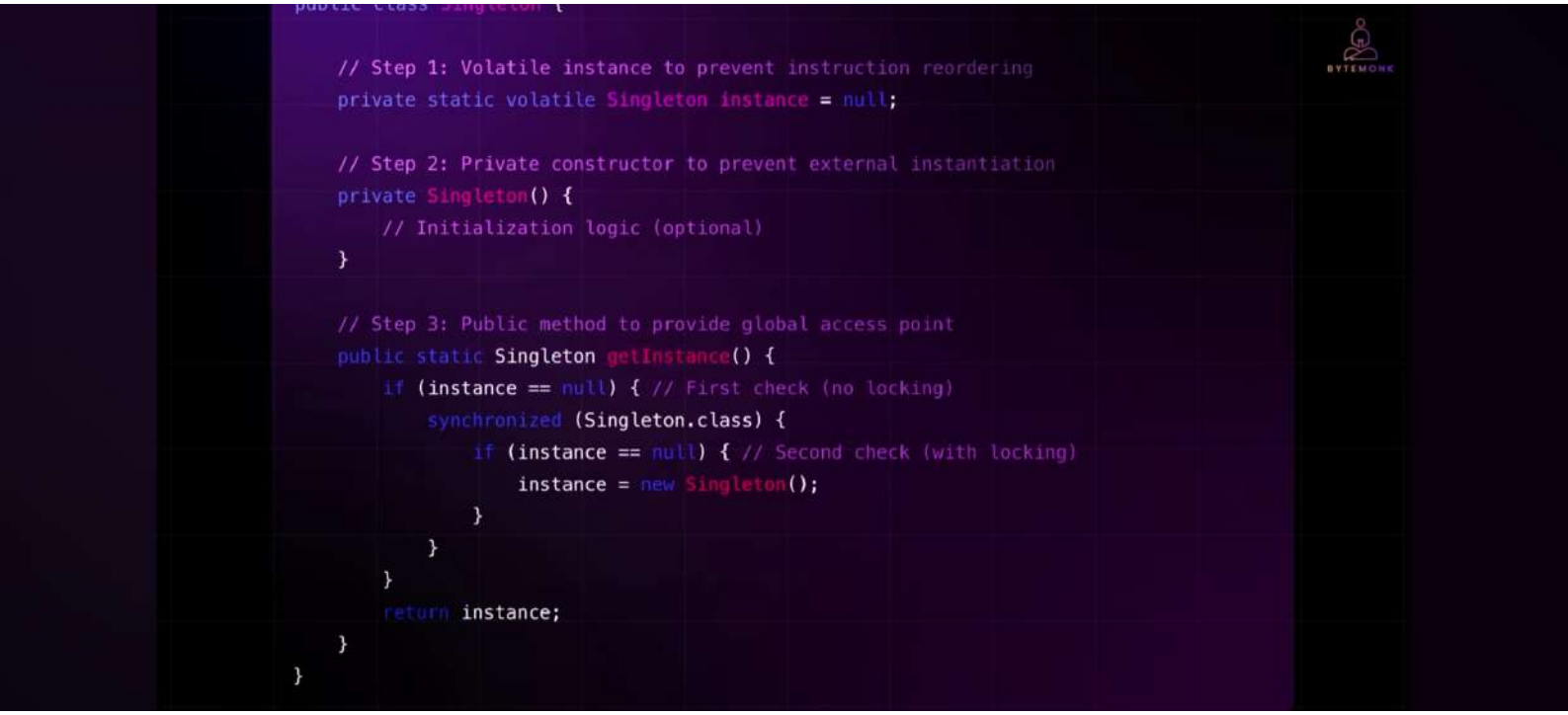
Double Check Locking

BYTEMONK

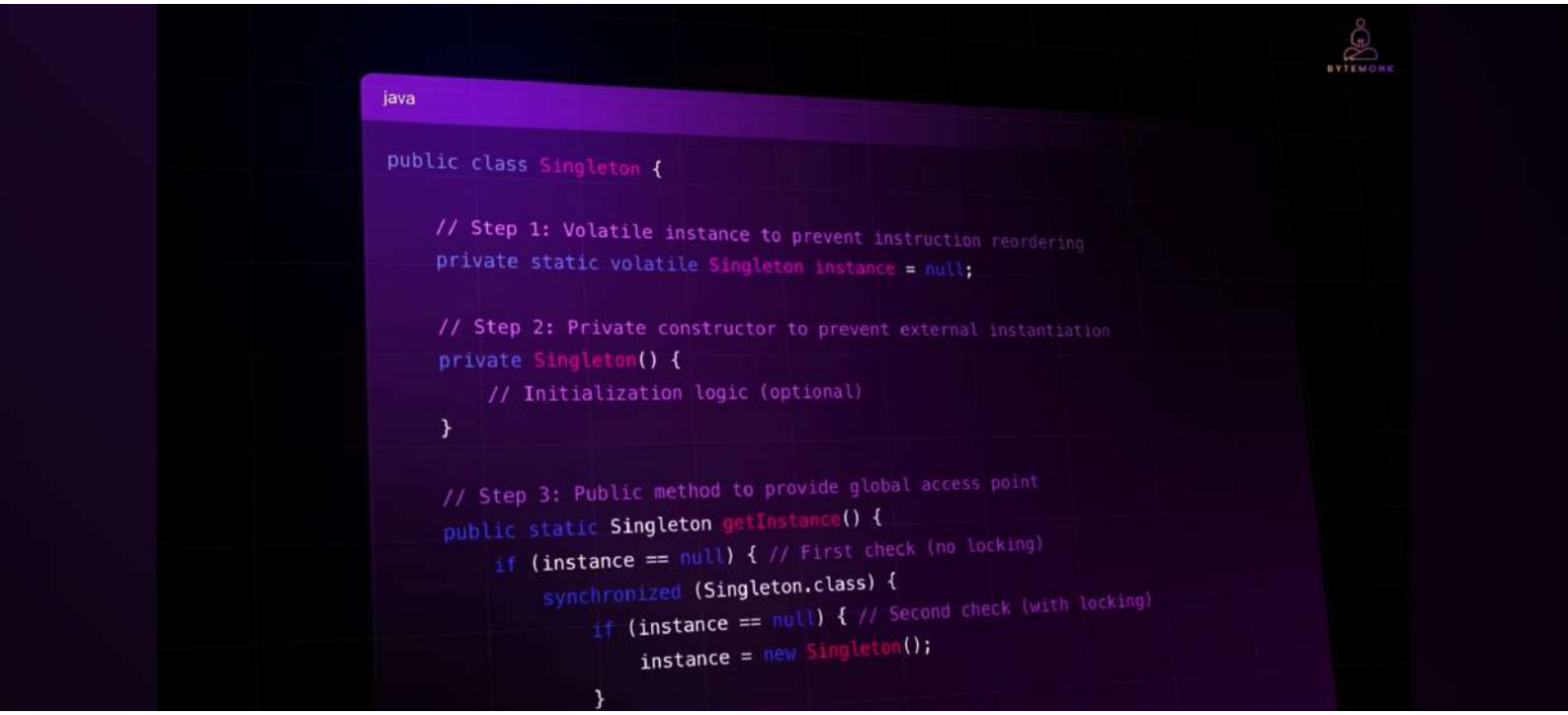
java

```java
public class Singleton {

    // Step 1: Volatile instance to prevent instruction reordering
    private static volatile Singleton instance = null;

    // Step 2: Private constructor to prevent external instantiation
    private Singleton() {
        // Initialization logic (optional)
    }

    // Step 3: Public method to provide global access point
    public static Singleton getInstance() {
        if (instance == null) { // First check (no locking)
            synchronized (Singleton.class) {
                if (instance == null) { // Second check (with locking)
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
}
```


// Step 2: Private constructor to prevent external instantiation
private Singleton() {
// Initialization logic (optional)
}
// Step 3: Public method to provide global access point
public static Singleton getInstance() {
if (instance == null) { // First check (no locking)
synchronized (Singleton.class) {
if (instance == null) { // Second check (with locking)
instance = new Singleton();
}
}
}
return instance;
}
}
BYTEMONK


public class Singleton {
// Step 1: Volatile instance to prevent instruction reordering
private static volatile Singleton instance = null;
// Step 2: Private constructor to prevent external instantiation
private Singleton() {
// Initialization logic (optional)
}
// Step 3: Public method to provide global access point
public static Singleton getInstance() {
if (instance == null) { // First check (no locking)
synchronized (Singleton.class) {
if (instance == null) { // Second check (with locking)
instance = new Singleton();
}
}
}
return instance;
}
}
BYTEMONK


BYTEMONK
// Step 1: Volatile instance to prevent instruction reordering
private static volatile Singleton instance = null;
// Step 2: Private constructor to prevent external instantiation
private Singleton() {
// Initialization logic (optional)
}
// Step 3: Public method to provide global access point
public static Singleton getInstance() {
if (instance == null) { // First check (no locking)
synchronized (Singleton.class) {
if (instance == null) { // Second check (with locking)
instance = new Singleton();
}
}
}
return instance;
}


BYTEMONK
java
public class Singleton {
// Step 1: Volatile instance to prevent instruction reordering
private static volatile Singleton instance = null;
// Step 2: Private constructor to prevent external instantiation
private Singleton() {
// Initialization logic (optional)
}
// Step 3: Public method to provide global access point
public static Singleton getInstance() {
if (instance == null) { // First check (no locking)
synchronized (Singleton.class) {
if (instance == null) { // Second check (with locking)
instance = new Singleton();
}


BYTEMONK
Singleton Pattern
Global Access Point


BYTEMONK
Singleton Pattern
Only One Instance

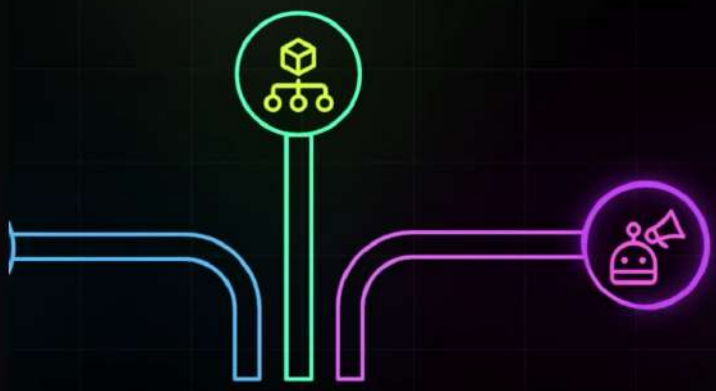
Singleton Pattern

BYTEMONK

- A configuration manager
- A logging service
- A thread pool

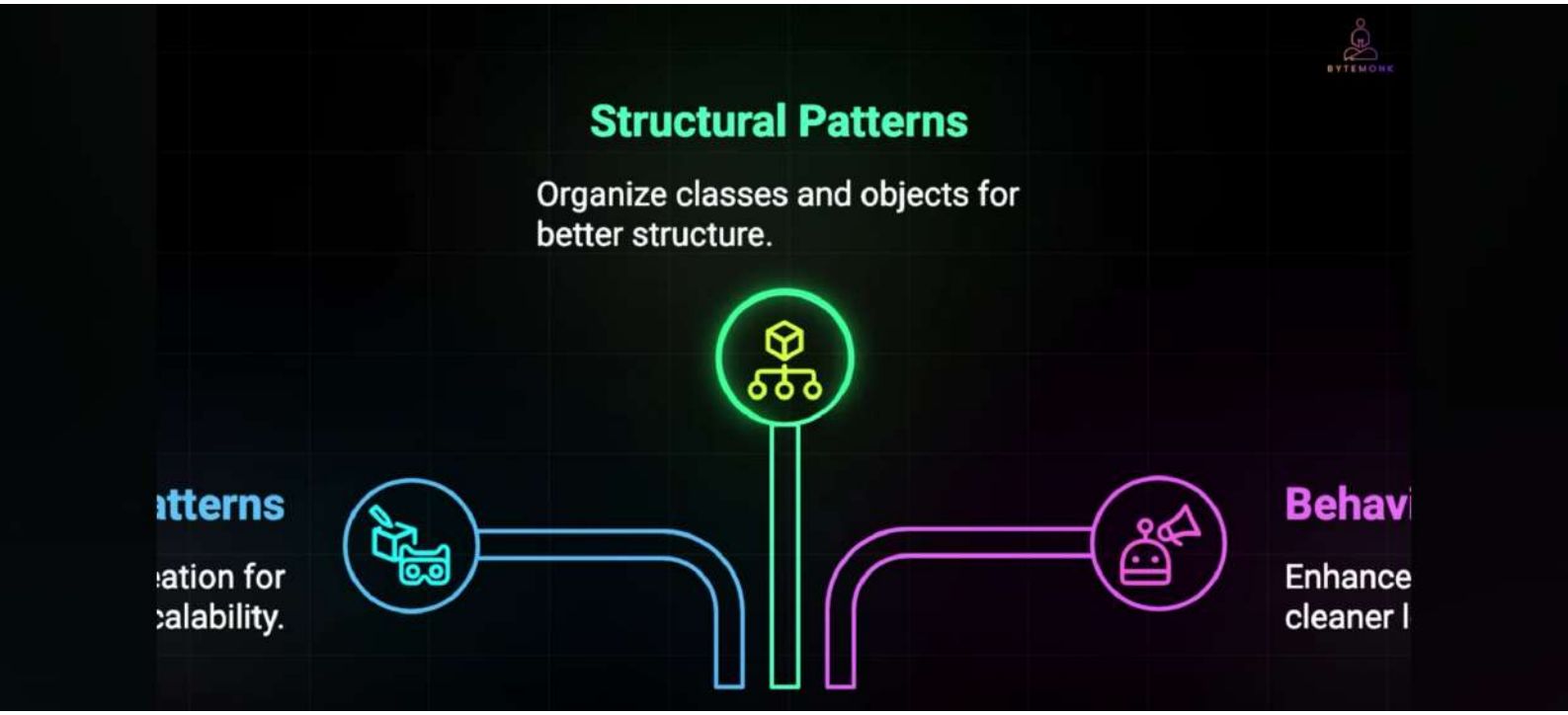

BYTEMONK
Singleton Pattern

Organize classes and objects for better structure.



Behavioral Patterns

Enhance object communication for cleaner logic.

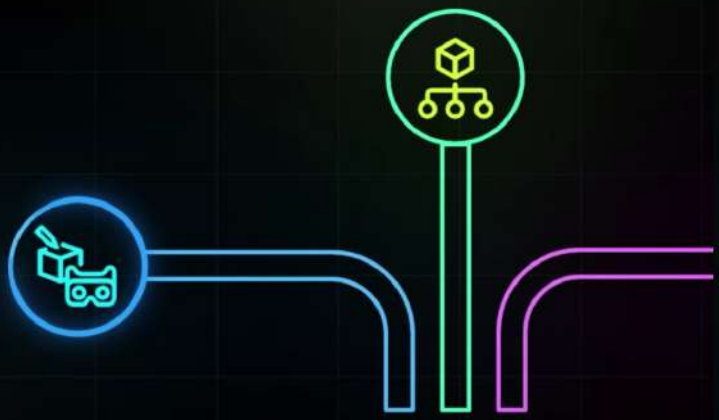

BYTEMONK
Structural Patterns
Organize classes and objects for better structure.
atterns
ation for
calability.
Behavi
Enhance
cleaner l

Organize classes and objects for better structure.

BYTEMONK

Creational Patterns

Focus on object creation for flexibility and scalability.



BYTEMONK

Structural Patterns

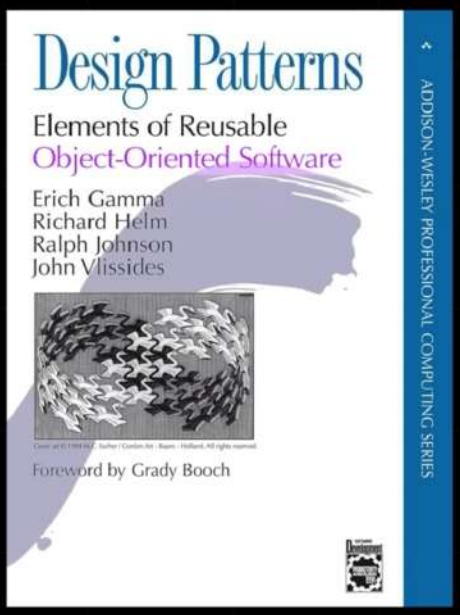
Organize classes and objects for better structure.

Creational Patterns

Focus on object creation for flexibility and scalability.

Behavioral Patterns

Enhance object communication for cleaner logic.


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

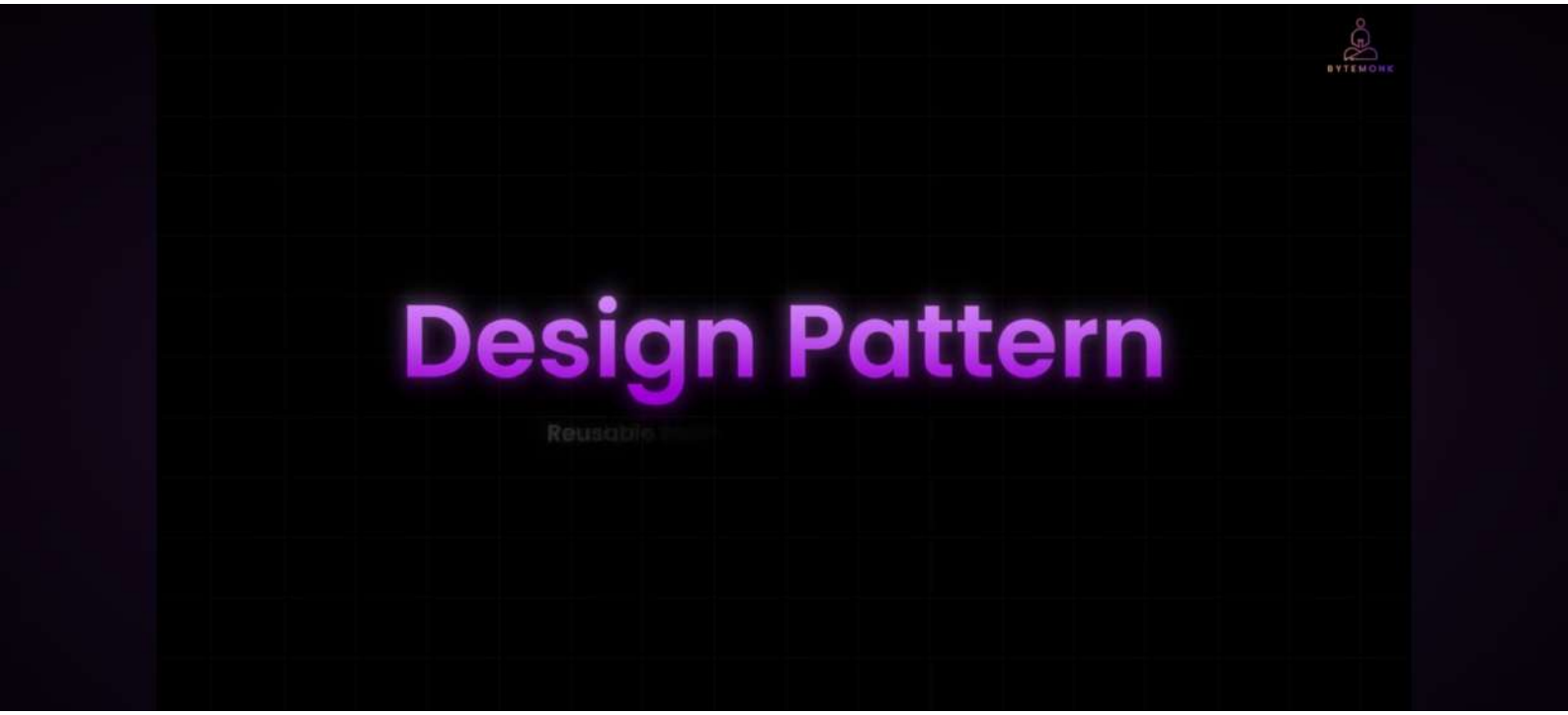
Gang of Four

John Vlissides

Erich Gamma

Ralph Johnson

Richard Helm


BYTEMONK
Design Pattern

BYTEMONK

Structural Patterns

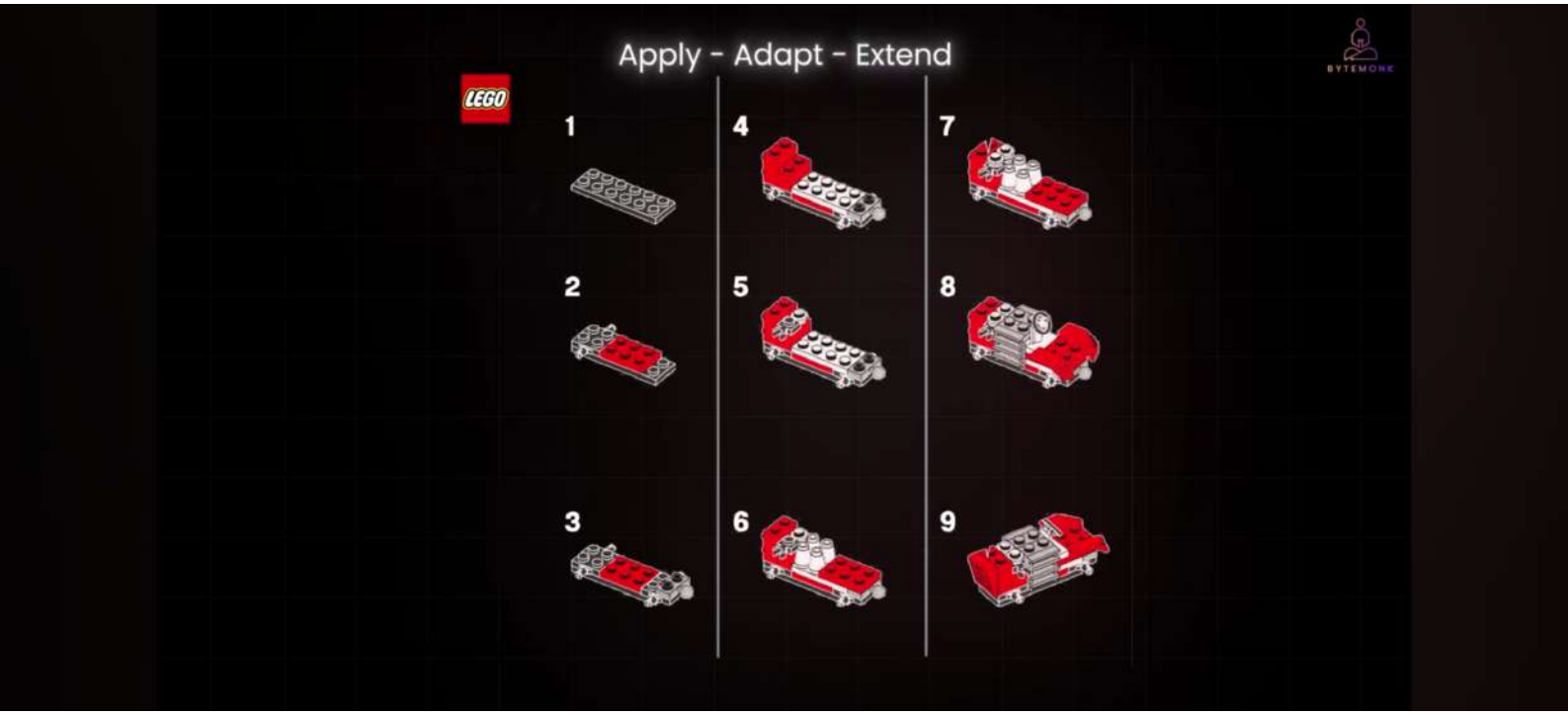
Organize classes and objects for better structure.

Creational Patterns

Focus on object creation for flexibility and scalability.

Behavioral Patterns

Enhance object communication for cleaner logic.


Apply - Adapt - Extend
BYTEMONK
1
2
3
4
5
6
7
8
9